REMARKS

Bearing in mind the comments in the Official Action, the application has been amended so as to place it in condition for allowance. An early indication of the same would be appreciated.

Claims 1-37 are now pending in this application. Claims 1, 11, 13, 22, 24, and 33 are independent. Claims 1, 11, 13, 22-24, and 33 have been amended, and dependent claims 35-37 have been added by this amendment.

Withdrawal of the objection to the Specification is requested. Responsive to the Examiner's objections, the "nonspecific statements" have been deleted or amended. Further, references to the conventional art discussed in the Specification have been clarified. Applicants submit that none of the cited Japanese applications are "essential material" for the purposes of this application, and that these references have been referred to in the Specification only in an attempt to apprise the Examiner of the conventional approaches available, prior to this application.

Withdrawal of the rejection of claims 1-34 under 35 U.S.C. §112, first paragraph, as not being enabled by the Specification, is requested.

The Examiner indicates that no algorithms are given and no analytical process is described, and that the specification makes reference to the claimed invention analyzing a computer program, and automatically generating program analysis information, but does not specifically explain the process. However, in the specification of the present invention, there is a description of the processes executed by the program analysis means, such as call graph and flow graph which are graphically displayed, influence range analysis, structure analysis and data flow anomaly analysis.

Applicants submit that, since each of these conventional processes are generally known, the specification of the present invention is not required to explain these processes in detail. Although metrics information, redundancy information and maintenance document information are generated as a result of a batch process, the methods of generating information are also not essential material in the disclosure of the present invention.

In addition, Applicants submit that it is common knowledge to those with skill in the art that metrics information, redundancy information and maintenance document information are generated on the basis of program analysis information, such as source code, syntactic analysis tree, symbol table, call graph, flow graph, data flow information, program dependence graph, and module I/O information.

Accordingly, Applicants submit that a person with skill in the art can fully understand and appreciate the novel and non-obvious features of the recited invention from the description contained in the present specification.

Further, Applicants respectfully traverse the Examiner's assertion that the claimed invention is not supported by either a concrete and tangible utility, or a well-established utility.

"Where statements in a disclosure concerning utility are not contrary to generally accepted scientific principles and the Examiner has not presented reasons to doubt their objective truth, the Examiner's unsupported skepticism as to the utility of the claimed invention does not provide a legally acceptable basis for rejecting the claims."[1]

Applicants invite the Examiner's attention to the disclosed utility as presented in the Specification.

For example, the utility of Applicants' invention may be found in the first paragraph on page 1 of the Specification, i.e., "[t]he present invention relates to a software analysis apparatus and method for analyzing a computer program to help the user easily understand the contents of the computer program, and a recording medium that stores a program for implementing such functions."

Further, the first and second paragraphs of the "SUMMARY OF THE INVENTION" also provides the utility of the invention, i.e., "[t]he present invention...has as its object to reliably

[1] *Ex parte Krenzer*, 199 U.S.P.Q. 227, 229 (PTO Bd. App. 1978).

analyze a large-scale program even when the memory capacity that can be mounted on a computer is limited...[and has, as] another object of the present invention[,] to efficiently use analysis information obtained before interruption even when analysis is interrupted due to some cause during program analysis."

Finally, the utility of Applicants' invention is also shown in the last paragraph on page 26 of the Specification, i.e., "[w]hen some analysis process is to be interactively done after generation of analysis information, since analysis information already stored in the database can be directly used, the time required for generating that analysis information can be omitted, and the wait time of the operator can be reduced compared to a conventional system which generates analysis information from the beginning every time new analysis process is started."

Applicants submit that the claimed invention has a clearly disclosed, tangible utility, and that the disclosure of conventionally known software analysis techniques is not necessary, as these features do not have a bearing on the novelty or non-obviousness of the recited invention. Therefore, the rejection for lack of enablement under §112 must be removed.

Withdrawal of the rejection of claims 1-34 under 35 U.S.C. §102(a) and 35 U.S.C. §102(e) as being anticipated by Wygodny et al. (USP 6,282,701) is rejected.

First of all, a rejection under §102(a) is not available in the rejection of these claims, because the invention of Wygodny et al. was not "known or used by others" in the sense required under §102(a), which requires that the disclosure of Wygodny et al. "...was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, ***before*** the invention thereof by the applicant for patent".[2] Under U.S. patent law, Wygodny et al. was "known" on the date it was published or issued as a patent, i.e., August 28, 2001, a date ***after*** Applicants' invention and filing date.

As for the anticipation rejection under 35 U.S.C. §102(e), Applicants note that anticipation requires the disclosure, in a prior art reference, of each and every limitation as set forth in the

[2] *See* 35 U.S.C. §102(a) (emphasis added).

claims.[3] There must be no difference between the claimed invention and reference disclosure for an anticipation rejection under 35 U.S.C. §102.[4] To properly anticipate a claim, the reference must teach every element of the claim.[5] "A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference".[6] "The identical invention must be shown in as complete detail as is contained in the ...claim."[7] In determining anticipation, no claim limitation may be ignored.[8]

In view of the foregoing authority, the cited reference at least fails to anticipate independent claims 1, 11, 13, 22, 24, and 33, as amended.

By way of background, a discussion of a preferred embodiment of the present application is provided below, followed by a discussion of the applied art, and distinctions of the disclosed and recited invention with respect to the applied art.

When a large number of programmers develop a large-scale program in collaboration or maintain a program already in existence, it is often difficult for a given programmer to understand a program written by other programmers. The present invention has been made to provide various kinds of information, such as call graph and flow graph, by automatically analyzing program source code, in order to help understanding of program, especially for analyzing a large-scale program.

Information available for analyzing a program (analysis information) generally includes a syntactic analysis tree, a symbol table, call graph, flow graph, data flow information, program dependence graph, module I/O information, for example. However, the various techniques for obtaining this type of information is generally known in the art and, consequently, obtaining this information is not considered "essential material" in the present invention.

---

[3] *Titanium Metals Corp. v. Banner*, 227 USPQ 773 (Fed. Cir. 1985).
[4] *Scripps Clinic and Research Foundation v. Genentech, Inc.*, 18 USPQ2d 1001 (Fed. Cir. 1991).
[5] *See* MPEP § 2131.
[6] *Verdegaal Bros. v. Union Oil Co. of Calif.*, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987).
[7] *Richardson v. Suzuki Motor Co.*, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989).
[8] *Pac-Tex, Inc. v. Amerace Corp.*, 14 USPQ2d 187 (Fed. Cir. 1990).

When a large-scale program is to be analyzed, the volume of analysis information required for program analysis also becomes very large, resulting in poor analysis efficiency. For example, the maximum memory capacity may be insufficient for all pieces of analysis information, or the time required for analysis may become very long.

The present invention has been developed in consideration of the above situation, and has, its objective, efficient analysis of large-scale programs, and to provide a software analysis apparatus and a software analysis method that allows efficient use of the analysis information obtained, even when analysis is interrupted due for some reason during program analysis.

In order to achieve the above objects, the present invention is characterized by sequentially storing the program analysis information in a predetermined data recording medium in an arbitrary unit or at an arbitrary timing, as claimed in the pending claims.

This approach allows the present invention to have remarkable effects including safely obtaining program analysis information without causing any memory shortage in the middle of analysis, even when the memory capacity mounted on the computer is limited. Further, even when the analysis process is interrupted due to some problem, information stored in the data-recording medium up until the time of interruption is safely protected. Hence, the process is resumed from the point at which it was interrupted, thus avoiding repetitive processes and greatly improving analysis efficiency.

Furthermore, when interactive analysis processing is done after the generation of analysis information, since analysis information already stored in the database can be directly used, the time required for generating the previously obtained analysis information can be omitted, so that the wait time of the operator can be reduced, compared to a conventional system which generates analysis information from the beginning, every time a new analysis process is started.

In contrast, Wygodny et al. facilitates the process of tracing the program execution paths, and relates to the process of monitoring and analyzing the execution of computer programs during the debugging process. In Wygodny et al., the tracing is performed without requiring

modifications to the executable modules or source code, and the trace data is collected according to instructions in a trace control dataset. Wygodny et al. propose a software system that facilitates the process of identifying and isolating bugs within a client program by allowing a developer to trace the execution paths of the client.

With reference to the general differences in the present invention, and with particular respect to the recited distinctions of pending claims 1-10 of the present invention over Wygodny et al., and considering the Examiner's comment that Wygodny et al. disclose a program analyzer method and apparatus having a GUI (Graphical User Interface) which collects trace information for use in analyzing a computer program, Applicants point out that "trace", as used in Wygodny et al., and "analysis information" in the present invention are different from each other.

The present invention has the feature that the analysis information is classified, and stored in a data-recording medium using a hierarchical structure, considering that the large volume of analysis information can be generated when a large-scale program is analyzed.

On the other hand, the trace data as in Wygodny indicates values of executed functions and variable values when the program is executed. The volume of trace data may become large if the time required for executing programs is long, or if a large number of instructions are executed.

Further, the trace data of Wygodny et al. is trace information generated by a debugger, and generated with the executing program. The trace data of Wygodny et al. is very different from the program analysis information recited and disclosed by the present invention, such as a syntactic analysis tree, call graph and flow graph, as further discussed below.

Generally, a debugger examines how variable values change as a program executes. The program analysis information of the present invention, such as syntactic analysis tree, call graph, and flow graph, is statically obtained from source code, but is not generated during execution of the program.

The Examiner has also indicated that trace data is collected according to instructions in a trace options file that is set up through an interactive process by an operator. However, as has been already discussed, trace data and analysis information are technically different from each other. In the trace file of Wygodny et al., for example, a user defines previously analysis objectives, and then the user can view the trace results after the execution of tracing.

On the other hand, in the present invention, it is not predetermined how to analyze by an interactive process. The program analysis means read out the program analysis information from the data recording medium, and a user performs an interactive process, for example, the user instructs program analysis units through GUI to execute a process as viewing the analysis results, and then the program analysis is executed in accordance with the results of the interactive process. That is, program analysis information is generated based on execution of the program, and a user can decide what to look for later. In this way, functions and effects are different from the interactive process in the present invention, with respect to the settings applied to the trace options file in Wygodny et al.

Although the Examiner has indicated that the trace data provides a graphical representation of the collected (analyzed) data showing program flow, program calls, and I/O information in Figs. 3A, 3B, 5 to 7, 13 and 14, Wygodny et al. does not disclose any of a flow graph, a call graph or I/O information corresponding to the program analysis information of the present invention.

The call graph in the present invention shows a calling relationship among functions, whereas Fig. 3A of Wygodny et al. shows actual calls and returns of functions in a tree structure. Applicants submit that these approaches are fundamentally different from each other.

The module I/O in the present invention determines I/Os of a function as arguments, global variables, or system functions, whereas that of Wygodny et al. determines "a value" of a variable, such as arguments when a function is executed.

In addition, the influence range analysis in the present invention (or in general program analysis) analyzes the range of influence imposed when a certain location (one instruction or one variable) in a program has been changed. The influence range is obtained by checking the relationship of variables in source code of the program. In Wygodny et al., tracking of variables is accomplished during execution of programs, therefore, it is different from the range instructing approach in the present invention.

As described above, since the recited program analysis, including classification of the analysis information as recited in claim 1 of the present invention is different from the tracing performed by the debugging tool in Wygodny et al., to regard the program analysis information of the present invention as being the same as the trace data in Wygodny et al., would be clearly erroneous.

In summary, and with reference to the invention claimed in claims 1-10, Wygodny et al. does not disclose, among other features, "program analysis information storage means for classifying the program analysis information generated by said program analysis information generation means in an arbitrary unit or at an arbitrary timing, and sequentially storing the program analysis information in a predetermined data recording medium...", as recited in independent claim 1, as amended.

Therefore, Applicants submit that, because of the above-noted deficiency, Wygodny et al. cannot settle the problem of poor analysis efficiency, since the time required for analysis becomes very long when a large-scale program is to be analyzed.

Accordingly, withdrawal of the anticipation rejection and allowance of independent claim 1 are requested. Further, as dependent claims 2-10 ultimately depend from claim 1 and, consequently incorporate its allowable features, these claims are also submitted as being allowable, at least on the basis of the allowability of the independent claim, as well as in their own right.

With reference to claims 11-12, the Examiner also indicated that Wygodny et al. discloses a program analyzer that provides a graphical representation of the collected (analyzed) data showing program flow (hierarchy), program calls, and I/O information. The analyzed data is stored in a database that can be read in method that is predetermined the operators inputs.

However, as discussed above, program flow (hierarchy), program calls and I/O information of Wygodny are different from the program analysis information of the present invention.

Although the Examiner considers "program flow" as "hierarchy", it is known to persons having skill in the art that program flow does not indicate a hierarchical structures, and "hierarchy" as claimed in claims 11 and 12 of the present invention indicates a structure for registering the generated program analysis information, which is shown in, for example, Fig. 2. Accordingly, "hierarchy" recited in claims 11 and 12 of present invention has nothing to do with the "program flow" in Wygodny et al.

As for "predetermined data" indicated by the Examiner, in the present invention, the hierarchical structure registering the generated program analysis information is predetermined, but the operator inputs are not predetermined. Further, Wygodny et al. does not disclose that program analysis information is stored in a database, and that the database is an object-oriented database.

Applicants submit that Wygodny et al. is at least deficient with respect to disclosing "...means for hierarchically registering the generated program analysis information in a database in units of analysis objectives", as recited in independent claim 11. Accordingly, reconsideration and allowance of independent claim 11 and dependent claim 12 are solicited.

With respect to the software analysis method of claims 13 to 21, similar arguments hold as for the above-discussed apparatus claims 1-10, particularly with respect to the differences between the trace data in Wygodny et al. for use in analyzing a computer operation, and the program analysis information of the present invention. In addition, Wygodny et al. does not

disclose any of metrics information, redundancy information, data flow anomaly information, or maintenance document information.

In particular, Wygodny et al. at least does not disclose, "classifying the program analysis information in an arbitrary unit or at an arbitrary timing...", as recited in amended claim 13. Accordingly, claims 13 to 21 should not be rejected under 35 U.S.C. 102.

As for the software analysis method of claims 22 and 23, and for arguments similar to those presented above with respect to claims 11 and 12, Wygodny et al. is at least deficient with respect to disclosing "hierarchically registering the generated program analysis information in a database in units of analysis objectives...", as recited in independent claim 22. Accordingly, reconsideration and allowance of claims 22-23 are requested.

As for the computer readable storage medium of claims 24-32, and for reasons similar to those discussed above, Wygodny et al. is at least deficient with respect to disclosing "a program analysis information classification function of classifying the program analysis information generated by the program analysis information generation function in an arbitrary unit or at an arbitrary timing", as recited in independent claim 24, as amended. Accordingly, reconsideration and allowance of claims 24-32 are requested.

As for the computer readable storage medium of claims 33-34, and for reasons similar to those discussed above, Wygodny et al. is at least deficient with respect to disclosing "hierarchically registering the generated program analysis information in a database in units of analysis objectives", as recited in independent claim 33. Accordingly, reconsideration and allowance of claims 33-34 are requested.

New dependent claims 35-37, depending, respectively, from independent claims 1, 13, and 24, have been drafted to avoid the applied art, and to further define the claimed invention. These claims are submitted as being patentable in their own right, as well as on the patentability of their respective base claims. Consideration and allowance of these claims is also requested.

Various ones of the pending claims have also been amended for clarity, to place them in a form more customary for U.S. practice.

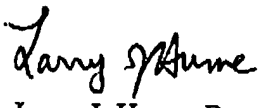
In view of the above, consideration and allowance of pending claims 1-37 are, therefore, respectfully solicited.

In the event the Examiner believes an interview might serve to advance the prosecution of this application in any way, the undersigned attorney is available at the telephone number noted below.

Pursuant to 37 C.F.R. §§1.17 and 1.136(a), Applicant hereby petitions for an extension of time for two (2) months to April 8, 2002 for filing a reply to the Office Action dated November 7, 2001 in connection with this application. The petition and deposit account authorization is enclosed herewith.

The Director is hereby authorized to charge any fees, or credit any overpayment, associated with this communication, including any extension fees, to CBLH Deposit Account No. 22-0185.

Respectfully submitted,



Larry J. Hume, Reg. No. 44,163
Customer Number 30678
Connolly Bove Lodge & Hutz LLP
1990 M Street, N.W., Suite 800
Washington, D.C. 20036-3425
Telephone: 202-331-7111

Enclosure: Specification after Amendment
Claims after Amendment
Amendment Transmittal Document (with claim fee calculation)
Petition for two (2) month extension of time

Specification after Amendment

IN THE SPECIFICATION:

Please replace the third paragraph starting at line 21 on page 1, and continuing through line 2 of page 3 of the Specification with the following replacement paragraph:

Conventionally, an apparatus which provides various kinds of information by automatically analyzing program source code ~~and helps~~to help understand a program has been proposed. For example, ~~an~~ a conventional apparatus and method for generating a call graph or flow graph by analyzing program source code to allow a programmer to visually recognize the program have been proposed. ~~Such apparatus and method~~ and are disclosed in, e.g., Japanese Patent Application Nos. 9-32415, 9-32452, ~~and the like already~~previously filed by the present applicant.

Please replace the third full paragraph starting at line 19 on page 4 of the Specification with the following replacement paragraph:

Furthermore, analysis may be interrupted due to some cause other than memory shortage during program analysis. In such case, ~~all~~ the analysis results previously obtained ~~so far~~ cannot be used, and program analysis must be redone from the beginning, resulting in poor analysis efficiency.

Please replace the second full paragraph starting at line 19 on page 9 and continuing through line6, page 10 of the Specification with the following replacement paragraph:

In this embodiment, program analysis information contains a syntactic analysis tree (including a symbol table), call graph, flow graph, data flow information, program dependence graph, module I/O information, metrics information, redundancy information, maintenance document information, ~~and the like~~for example, as shown in Fig. 2. Of ~~these~~ this information, the syntactic analysis tree, call graph, flow graph, data flow information, program dependence graph, module I/O information, ~~and the like~~for example, are generated by the program analysis

information generation unit 11 shown in Fig. 1, while the metrics information, redundancy information, maintenance document information, ~~and the like~~for example, are generated by a program analysis unit 20_i, shown in Fig. 3, that analyzes a program by a batch process.

Please replace the first full paragraph starting at line 7, page 10 of the Specification with the following replacement paragraph:

Note that the "symbol table" includes a group of information that represent meanings ~~and the like~~ of symbols (variables) used in a program. For example, when character "a" is used at different locations in a program, they may indicate identical variables or different variables. For example, when variables expressed by an identical character are used in two different functions, if those variable are global variables, they are identical ones; if the variables are locally defined in the individual functions, they are different ones.

Please replace the first and second paragraphs starting at line 7, page 12 of the Specification with the following replacement paragraphs:

The "data flow information" is information as a result of analysis of the data flow such as alias information of a pointer, definition-use chain information that indicates the use location of a variable defined at a given location, ~~and the like~~for example. Using this data flow information and ~~the like~~similar information, for example, a data flow anomaly can be inspected. The data flow anomaly is a combination of illegal events with respect to data. All data must be used while keeping given rules, i.e., each data is used after it is defined, and is undefined finally. An illegal combination that does not keep such rules causes data flow anomaly.

Even when data flow anomaly is present, source code can be compiled. However, upon executing the compiled program in practice, data flow anomaly may appear when control takes an anomalous path. Hence, such anomaly is preferably removed in advance in the process of source code. ~~As for data~~Data flow anomaly detection~~, refer to patent applications (~~ is known in the art, as disclosed in Japanese Patent Application Nos. 9-32415 and 9-32452~~),~~ by the present applicant.

Please replace the third paragraph starting at line 15, page 13 of the Specification with the following replacement paragraph:

The "metrics information" pertains to numeration indices of software. The software analysis apparatus of this embodiment measures metrics which represent quantitative complexity, and those which represent qualitative complexity, and generates ~~their~~ respective information. As metrics of quantitative complexity, two different kinds of quantities, i.e., a size metric that measures the physical description quantity of a program, and cyclomatic number that measures the complexity of a control structure are measured. On the other hand, ~~as metrics of qualitative complexity,~~ module cohesion and module coupling ~~that express~~ relating to the contents of modules are measured, and are representative of the qualitative complexity of the software.

Please replace the second paragraph starting at line 9, page 14 of the Specification with the following replacement paragraph:

The "maintenance document information" is a document group used upon maintaining a program. More specifically, this information describes lists of procedures, types, variable names~~, and the like~~ defined in a program, and how the individual procedures are related ~~and the like~~, for example. This "maintenance document information" is used when a given programmer wants to understand a program created by another programmer. By acquiring the "maintenance document information", information can be provided in the hypertext format, and is ~~greatly~~ much more convenient compared to paper-based documents.

Please replace the second full paragraph starting at line 17, page 16 of the Specification with the following replacement paragraph:

By contrast, in the software analysis apparatus of this embodiment, the program analysis information storage unit 12 classifies program analysis information generated by the program analysis information generation unit 11 in units of kinds of analysis information in the respective

layers, and sequentially stores them as a database on the information recording medium 13 such as a hard disk ~~or the like~~, for example.

Please replace the first full paragraph starting at line 3, page 18 of the Specification with the following replacement paragraph:

Note that this embodiment uses as a database object-oriented database software, and generates program analysis information using an object-oriented language, e.g., C++ ~~or the like~~, for example. Such database is used for the following reason. That is, since the program analysis information has a complicated structure and relation such as a graph structure ~~and the like~~, if a relational database is used, it is hard to store information. Also, use of such a database is convenient, since program analysis information generated as an object on C++ can be stored in a structure as it is generated.

Please replace the paragraph starting at line 22, page 19 and continuing through line 6, page 20 of the Specification with the following replacement paragraph:

The program analysis unit 20_i executes an analysis process that can analyze by a batch process using various kinds of program analysis information stored in the information recording medium 13 by the program analysis information storage unit 12, and stores the process results as a batch analysis result file in the information recording medium 13 again. During this batch process, the operator need not input any instructions, and the apparatus of this embodiment automatically analyzes. The batch analysis result file contains, e.g., metrics information that pertains to the program scale, redundancy information, maintenance document information~~, and the like~~.

Please replace the paragraph starting at line 3, page 22 of the Specification with the following replacement paragraph:

Other examples of the analysis process done by the program analysis units 20 are structure analysis, data flow anomaly analysis, ~~and the like~~for example. In this embodiment, the

data flow anomaly analysis is done by an interactive process, but may be implemented by a batch process.

Please replace the paragraph starting at line 18, page 24 and continuing through line2, page 25 of the Specification with the following replacement paragraph:

A computer readable recording medium of the present invention will be described below. The software analysis apparatus of the present invention is implemented when the computer operates in accordance with a program stored in its ROM, or RAM, ~~or the like~~for example. Alternatively, the software analysis apparatus may be implemented when a computer that loads an external program operates according to that program. Hence, when a computer mounts a recording medium that records such program, e.g., a floppy disk, CD-ROM, magnetooptical disk, magnetic tape, semiconductor memory, or the like, and loads the program therefrom, the present invention can be practiced on the computer.

Claims after Amendment

IN THE CLAIMS:

Please amend claims 1, 13, 22-24, and 33 as follows:

1. (Amended) A software analysis apparatus comprising:

program analysis information generation means for automatically generating program analysis information required for analyzing a computer program;

program analysis information storage means for classifying the program analysis information generated by said program analysis information generation means in an arbitrary unit or at an arbitrary timing, and sequentially storing the program analysis information ~~generated by said program analysis information generation means~~ in a predetermined data recording medium ~~in an arbitrary unit or at an arbitrary timing~~; and

program analysis means for executing program analysis by reading out the program analysis information from said data recording medium.

11. (Amended) A software analysis apparatus for generating program analysis information required for analyzing a computer program, comprising:

means for hierarchically registering the generated program analysis information in a database in units of analysis objectives~~, and~~;

means for implementing analysis of the hierarchically registered program analysis information by reading out the program analysis information already registered in a predetermined layer in correspondence with an analysis objective upon analyzing the computer program.

13. (Amended) A software analysis method, comprising:

~~the program analysis information generation step of~~ automatically generating program analysis information required for analyzing a computer program;

classifying the program analysis information in an arbitrary unit or at an arbitrary timing;

~~the program analysis information storage step of~~ sequentially storing the program analysis information ~~generated in the program analysis information generation step~~ in a predetermined data recording medium ~~in an arbitrary unit or at an arbitrary timing~~; and

~~the program analysis step of~~ executing program analysis by reading out the program analysis information from said data recording medium.

22. (Amended) A software analysis method for generating program analysis information required for analyzing a computer program, comprising:

hierarchically registering the generated program analysis information in a database in units of analysis objectives, and

implementing analysis by reading out the program analysis information already registered in a predetermined layer in correspondence with an analysis objective upon analyzing the computer program.

23. (Amended) A method according to claim ~~21~~22, wherein the database is an object-oriented database.

24. (Amended) A computer readable recording medium recording a computer program for making a computer implement:

a program analysis information generation function of automatically generating program analysis information required for analyzing a computer program;

a program analysis information classification function of classifying the program analysis information generated by the program analysis information generation function in an arbitrary unit or at an arbitrary timing;

a program analysis information storage function of sequentially storing the classified program analysis information generated by the program analysis information ~~generation~~ classification function in a predetermined data recording medium ~~in an arbitrary unit or at an arbitrary timing~~; and

a program analysis function of executing program analysis by reading out the classified program analysis information from said data recording medium.

33. (Amended) A computer readable storage medium recording a program for making a computer implement a function of:

generating program analysis information required for analyzing a computer program,

hierarchically registering the generated program analysis information in a database in units of analysis objectives, and

implementing analysis by reading out the program analysis information already registered in a predetermined layer in correspondence with an analysis objective upon analyzing the computer program.